Visualisierung von IFC-Objekten mittels Java3D

Michael Theiler [1], Eike Tauscher [1], Jan Tulke [2], Thomas Riedel [3]

[1] *Informatik im Bauwesen, Bauhaus-Universität Weimar*
{michael.theiler | eike.tauscher}@uni-weimar.de
[2] *HOCHTIEF Construction AG*
jan.tulke@hochtief.de
[3] *Informations- und Wissensverarbeitung, Bauhaus-Universität Weimar*
thomas.riedel@uni-weimar.de

Kurzfassung

Die Planung komplexer Bauwerke erfolgt zunehmend mit rechnergestützten Planungswerkzeugen, die den Export von Bauwerksinformationen im STEP-Format auf Grundlage der Industry Foundation Classes (IFC) ermöglichen. Durch die Verfügbarkeit dieser Schnittstelle ist es möglich, Bauwerksinformationen für eine weiterführende applikationsübergreifende Verarbeitung bereitzustellen. Ein großer Teil der bereitgestellten Informationen bezieht sich auf die geometrische Beschreibung der einzelnen Bauteile. Um den am Bauprozess Beteiligten eine optimale Auswertung und Analyse der Bauwerksinformationen zu ermöglichen, ist deren Visualisierung unumgänglich. Das IFC-Modell stellt diese Daten mit Hilfe verschiedener Geometriemodelle bereit. Der vorliegende Beitrag beschreibt die Visualisierung von IFC-Objekten mittels Java3D. Er beschränkt sich dabei auf die Darstellung von Objekten, deren Geometrie mittels Boundary Representation (Brep) oder Surface-Model-Repräsentation beschrieben wird.

Stichworte

IFC, Java3D, Brep, Surface-Model, STEP, 3D-Modell, Schnittstelle, Visualisierung

1 Einleitung

Mit den Industry Foundation Classes (IFC) existiert eine international genormte Schnittstelle für den Austausch von Bauwerksinformationen. Nahezu jede etablierte CAD-Planungssoftware ermöglicht den Export von IFC-Daten mittels STEP (Standard for the exchange of product model data). Durch diese Schnittstelle ist es möglich, Bauwerksinformationen für deren weiterführende Verarbeitung zu verwenden. Eine

Visualisierung von geometrischen Gebäude- oder Bauteilinformationen ist beispielsweise für eine nachvollziehbare Verknüpfung mit Bauprozessen im Sinne einer 4D-Simulation zwingend notwendig. Die IFC bieten diverse Möglichkeiten, um Geometriedaten sowohl für 2D-, als auch für 3D-Repräsentationen abzubilden. Das IFC-Modell beinhaltet ein multiples Darstellungskonzept, d.h. ein Objekt, z.B. eine Wand, kann mehrere Darstellungsarten besitzen. Die wichtigsten 3D-Darstellungen sind die Boundary Representation (Brep) und die Repräsentation als Extrusions- und Rotationskörper (SweptSolid). Hinzu kommen, vor allem bei den Extrusionskörpern, weitere Anforderungen hinsichtlich CSG-Modellierung (Constructive Solid Geometry), um beispielsweise Öffnungen für Türen und Fenster von einer Wand abzuziehen (herauszuschneiden) oder um Wände an einer bestimmten Ebene zu kappen (Clipping). Am Lehrstuhl Informatik im Bauwesen der Bauhaus-Universität Weimar wird innerhalb verschiedener Projekte an einem *IFC-Viewer* auf Basis der Programmiersprache Java gearbeitet. Dieser Beitrag beschreibt auszugsweise die Visualisierung von IFC-Objekten mittels Java3D am Beispiel von Brep- und Surface-Model-Repräsentationen. Die notwendigen Algorithmen und Methoden werden im Detail vorgestellt.

2 Geometrische Repräsentationen der IFC

In den IFC werden alle Objekte in eine vorgegebene räumliche Struktur eingebettet. Diese Struktur orientiert sich an der räumlichen Abbildung eines Gebäudes (*IfcSpatialStructureElement*). Jeder Ebene können Objekte mit einer geometrischen Repräsentation zugeordnet werden. Diese Objekte werden in den IFC durch Instanzen vom Typ *IfcProduct* repräsentiert. Ein *IfcProduct* ist dabei eine Superklasse für viele andere Klassen (z.B. *IfcWall*) und definiert neben einem eindeutigen Identifikator (*GlobalId*) für das Objekt zwei weitere wichtige Attribute in Bezug auf die geometrische Repräsentation. Dies ist zum einen die Positionierung im Raum (*ObjectPlacement*), zum anderen der geometrische Aufbau des Objekts (*Representation*). Das Attribut *ObjectPlacement* beschreibt die Lage und die Ausrichtung des lokalen Koordinatensystems für die Geometrie eines Objekts. Die Positionierung kann in den IFC auf zwei verschiedene Arten erfolgen. Die Klasse *IfcGridPlacement* ermöglicht die Ausrichtung des lokalen Koordinatensystems an einem Rastergitter. Alternativ wird für die Positionierung die Klasse *IfcLocalPlacement* benutzt. Dabei wird ein neues lokales Koordinatensystem definiert, indem eine Transformation (Translation und Rotation) in Bezug auf ein anderes Koordinatensystem angegeben wird. Dieses kann wiederrum lokal definiert sein oder es bezieht sich auf den globalen Koordinatenursprung (*Abb. 1*).

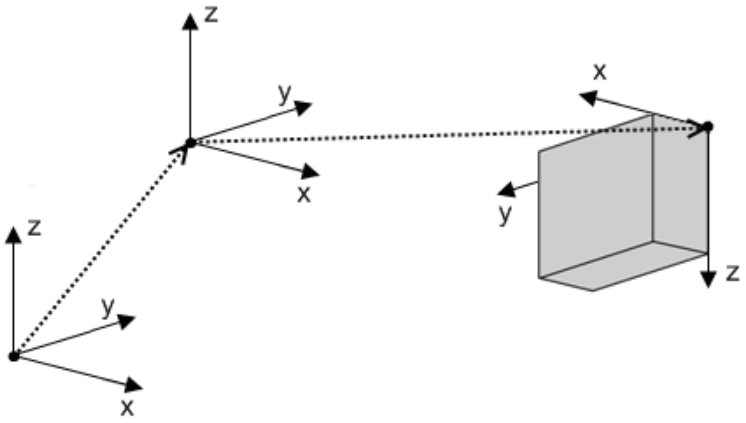

Abb. 1: Beispiel - Relatives Positionieren von lokalen Koordinatensystemen.

Durch das multiple Darstellungskonzept kann ein IFC-Objekt mehrere Repräsentationen besitzen. Folgende Darstellungsarten werden unterschieden:

- *2D-Darstellungen:* Curve2D, Annotation2D
- *3D-Oberflächenmodelle:* SurfaceModel
- *3D-Volumenmodelle:* SweptSolid, Brep, CSG, Clipping, BoundingBox
- *Mischformen:* GeometricSet, SectionedSpine, MappedRepresentation
- *Topologische Repräsentationen:* TopologyRepresentation

Jedes IFC-Objekt wird in ein oder mehrere Elemente zerlegt. Eine Treppe kann beispielsweise aus mehreren Teilen zusammengesetzt sein (Stufen, Geländer, etc.). Diese Teilelemente werden nachfolgend als *Items* bezeichnet und in den IFC durch die Klasse *IfcRepresentationItem* abgebildet. Die Klasse *IfcRepresentationItem* ist die Superklasse für alle Darstellungsarten innerhalb des IFC-Modells. Unterklassen davon sind beispielsweise *IfcManifoldSolidBrep* oder *IfcSweptAreaSolid*, die jeweils das notwendige Modell für ihre entsprechende Repräsentationsart abbilden. Weitere für die Visualisierung notwendige Informationen wie Farbe oder Reflexionsgrad werden mit der Klasse *IfcStyledItem* den *Items* zugewiesen.

3 Brep- und Surface-Repräsentationen der IFC

Das Boundary-Representation-Modell (Brep) beschreibt einen 3D-Körper durch die Definition seiner begrenzenden Oberflächen. Die Normalen der Oberflächen zeigen vom

Körper weg und definieren auf diese Weise die „Außenseiten“ des Körpers. Die Normale ergibt sich aus der Umlaufrichtung des Polygonzugs der Außenkante der jeweiligen Fläche. Außenkanten werden gegen den Uhrzeigersinn angegeben, Innenkanten (Löcher) entgegengesetzt [1].

Eine Brep-Repräsentation beschreibt demnach einen geschlossenen Körper. Für das Surface-Model ist diese Festlegung nicht definiert. Das heißt, ein Surface-Model beschreibt keinen geschlossenen Körper, sondern lediglich dreidimensionale Flächen. Beide Darstellungsarten werden vor allem für komplexe Geometrien verwendet, die sich nicht effektiv als Extrusions-, Rotations- oder CSG-Körper modellieren lassen. Sie finden beispielsweise bei Türen, Fenstern, Treppen, Möbeln oder Landschaftsmodellen Anwendung.

Die *Items* eines IFC-Objekts mit einer Brep-Repräsentation sind vom Typ der Klasse *IfcManifoldSolidBrep*, die eine abstrakte Superklasse für *IfcFacetedBrep* und *IfcFacetedBrepWithVoids* ist. Beide Untertypen besitzen eine geschlossene äußere Hülle. Der zweite Typ hat zusätzlich mindestens einen Hohlraum innerhalb der Außenhülle. Die Hohlräume dürfen sich nicht gegenseitig überschneiden und werden ebenfalls als geschlossene Hülle angegeben. Die Normalen der Oberflächen zeigen in den Hohlraum hinein. Durch die Forderung nach einer geschlossenen Hülle mit gerichteten Oberflächen können Körper mit Brep-Repräsentation für eine eventuelle CSG-Modellierung weiterverwendet werden. Auf Objekte mit Surface-Model-Repräsentation trifft dies nicht zu, da dort auch offene Hüllen zugelassen sind. Die *Items* eines IFC-Objekts mit dieser Art der Darstellung sind entweder vom Typ *IfcFaceBasedSurfaceModel* oder *IfcShellBasedSurfaceModel*. Ersteres enthält lediglich eine Aufzählung aller Oberflächen, während letzteres aus einer Menge von geschlossenen oder offenen Hüllen zusammengesetzt ist. Für eine korrekte Darstellung von IFC-Objekten mit Surface-Model-Repräsentation müssen Vorder- und Rückseite der Oberflächen sichtbar sein.

Offene und geschlossene Hüllen werden in den IFC durch die Klasse *IfcConnectedFaceSet* repräsentiert. Dies ist die Superklasse für *IfcClosedShell* und *IfcOpenShell*, wobei ein Körper mit Brep-Repräsentation, wie bereits beschrieben, lediglich aus geschlossenen Hüllen (*IfcClosedShell*) bestehen darf. Eine Hülle setzt sich aus mehreren Oberflächen zusammen. Diese Oberflächen sind vom Typ *IfcFace* und beinhalten eine Menge (Set) von Konturen vom Typ der Klasse *IfcFaceBound*. Die äußere Kontur ist vom Typ der Unterklasse *IfcFaceOuterBound*. Sind mehr als eine Kontur angegeben, stellen die weiteren Konturen Löcher in der Oberfläche dar (*Abb. 2*). Ein *IfcFaceBound* hat zwei Attribute. Über das Attribut *Bound* erhält man einen Verweis

auf den beschreibenden Polygonzug. Das Attribut *Orientation* gibt an, ob die Umlaufrichtung des Polygons gegen die Konvention verläuft [2],[3].

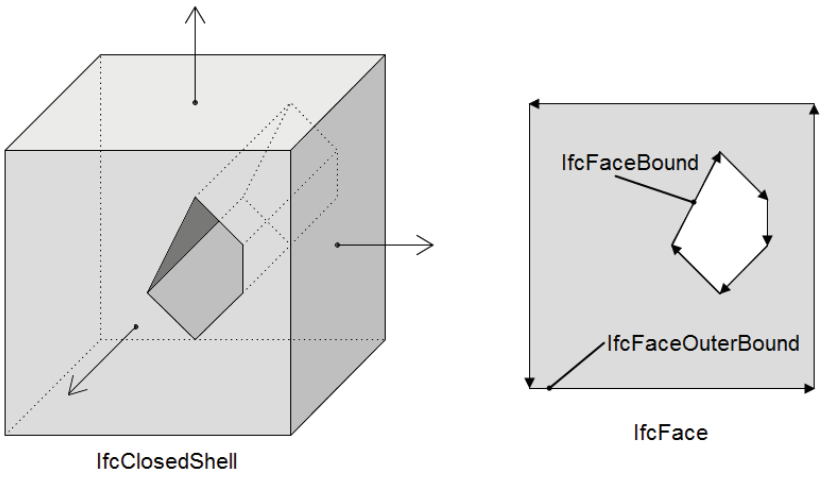

Abb. 2: IfcClosedShell und IfcFace.

4 Visualisierung mittels Java3D

Java3D ist eine leistungsfähige Erweiterung für die Standard Java API (Application Programming Interface) zur Darstellung von dreidimensionalen Grafiken basierend auf *OpenGL* sowie *DirectX*. Das interne Modell des Java3D API wird durch eine Baumstruktur beschrieben. Diese Struktur wird durch ein *Scene Graph* beschrieben [4]. Er stellt die topologischen Zusammenhänge der darzustellenden 3D-Geometrie und Transformationen graphisch dar. Für die Erzeugung des Java3D Modells werden für die gewählte Umsetzung folgende Klassen aus dem Java3D API benötigt:

- *BranchGroup:* Gruppe zur Aufnahme von Kindern
- *TransformGroup:* Gruppe zur Transformation aller Kinder
- *Switch:* Gruppe zum unsichtbar / sichtbar Schalten aller Kinder
- *Shape3D:* repräsentiert ein zu visualisierendes Objekt

Zu visualisierende Körper werden jeweils durch ein *Shape3D* Objekt repräsentiert. Die Klasse *Shape3D* besitzt für die Festlegung von Geometrie und Aussehen die Attribute *Geometry* und *Appearance* (Farbe, Reflexionseigenschaften, etc.). Für die Implementierung des *Scene Graphs* wurde eine neue Klasse *Object3D* entwickelt, die die Klasse *BranchGroup* durch Vererbung erweitert. Sie beinhaltet u.a. Methoden zur Aufnahme von *Shape3D* Objekten und zum Setzen der Transformation des gesamten

3D-Körpers. Die interne Struktur des *Scene Graphs* der Klasse *Object3D* ist in *Abb. 3* dargestellt. Dieser Aufbau ermöglicht es, allen Shapes eines Objekts eine gemeinsame Translation im Raum aufzutragen. Außerdem lassen sich so ganze Objekte mit ihren Teilelementen auf einfache Weise in der Darstellung ausblenden. Für jedes darzustellende IFC-Objekt wird ein neues Objekt der Klasse *Object3D* erzeugt. Da IFC-Objekte aus mehreren *Items* bestehen, wird für jedes *Item* ein neues *Shape3D* Objekt erzeugt, welches anschließend an die gemeinsame *TransformGroup* der *Object3D* Klasse gehangen wird. Das erzeugte *Object3D* wird dann schließlich an die Wurzel des *Scene Graphs* gehangen.

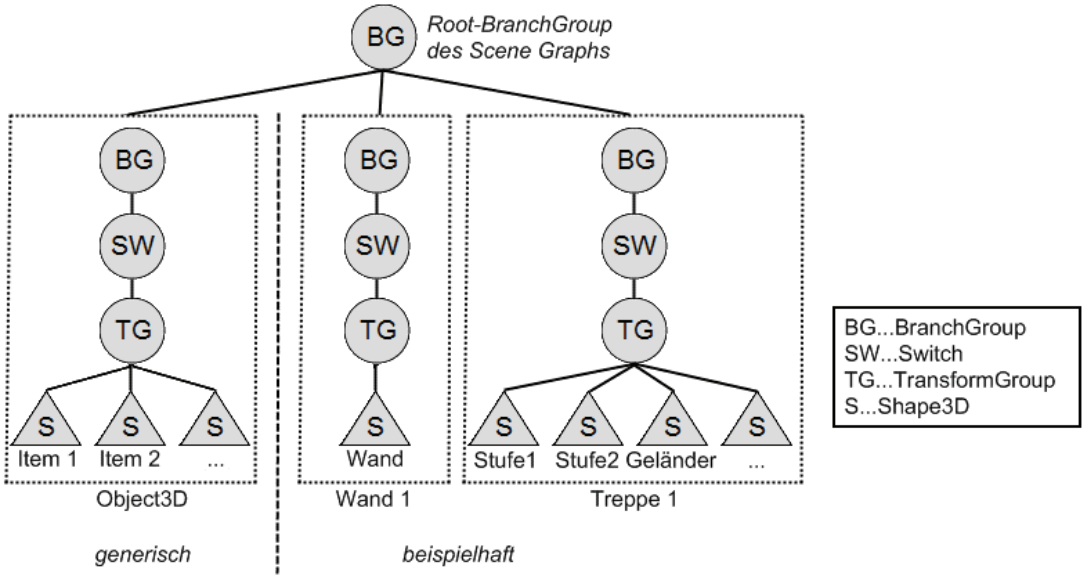

Abb. 3: Beispiel eines Scene Graphs.

Für die Darstellung der *Items* eines IFC-Objekts ist es erforderlich die Geometrie-Informationen der IFC in ein Objekt der Klasse *Geometry* aus dem Java3D API zu überführen. Die Utility-Klasse *GeometryInfo* aus dem Java3D API vereinfacht die Erzeugung der Geometrie und bietet diverse Möglichkeiten der Umwandlung u.a. in ein trianguliertes Oberflächenmodell. Zusammen mit der Klasse *NormalGenerator* lassen sich auf einfache Weise die Normalen der Oberflächen erzeugen, welche für die Berechnung von Lichtreflexen auf der Oberfläche benötigt werden. Diese Normalen haben keinen Einfluss auf die Bestimmung der Vorderseite, die ausschließlich aus der Umlaufrichtung des Polygonzugs der jeweiligen Fläche bestimmt wird.

Für die Erzeugung der Geometrie mittels der Utility-Klasse *GeometryInfo* werden die folgenden drei Felder benötigt:

- *int[] contourCounts:* gibt an, wie viele Konturen zu einer Oberfläche gehören. Die Länge des Feldes entspricht der Anzahl der darzustellenden Flächen. Die Werte des Feldes entsprechen der Anzahl der Konturen für die jeweilige Fläche.
- *int[] stripCounts:* gibt an, aus wie vielen Eckpunkten sich der Polygonzug der jeweiligen Kontur zusammensetzt. Die Länge des Feldes entspricht der Anzahl aller Konturen. Die Werte des Feldes entsprechen der Anzahl der Eckpunkte des jeweiligen Polygons.
- *Point3d[] vertices:* enthält die Eckpunkte der Konturen. Die Länge des Feldes entspricht der Anzahl aller Eckpunkte der Konturen.

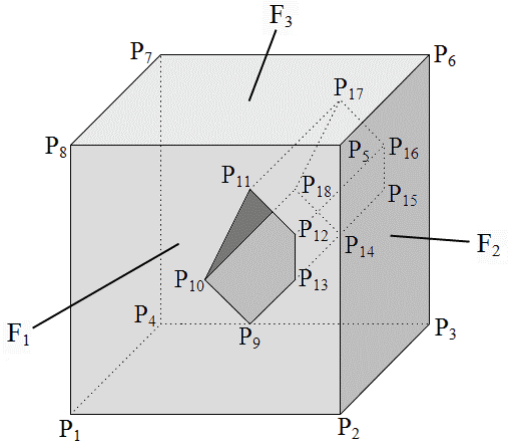

Abb. 4: Beispiel für die Visualisierung mit Java3D.

Zur Verdeutlichung wird die Erzeugung dieser Felder anhand eines Beispiels demonstriert. Für den Körper aus *Abb. 4* soll ein *Geometry* Objekt für die Darstellung mit Java3D erzeugt werden. Der Gesamtkörper V_1 besteht insgesamt aus elf Einzelflächen F_i.

$$V_1 = \{ F_1, F_2, F_3, F_4, \ldots , F_{10}, F_{11} \}$$

Die Flächen F_i setzen sich aus beliebig vielen Konturen K_j zusammen, wobei die erste Kontur jeweils die äußere Kante der Fläche F_i ist. Weitere Konturen beschreiben Polygonzüge von Löchern in der Oberfläche mit den Eckpunkten P_k.

$$F_1 = \{ K_1, K_2 \} \qquad K_1 = \{ P_1, P_2, P_5, P_8 \}$$

$$F_2 = \{ K_3 \} \qquad K_2 = \{ P_9, P_{10}, P_{11}, P_{12}, P_{13} \}$$

$$F_3 = \{ K_4 \} \qquad K_3 = \{ P_2, P_3, P_6, P_5 \}$$

$$F_4 = \{ K_5 \} \qquad K_4 = \{ P_5, P_6, P_7, P_8 \}$$

$$F_5 = \{ K_6 \} \qquad K_5 = \{ P_8, P_7, P_4, P_1 \}$$

$$F_6 = \{ K_7 \} \qquad \ldots$$

$$\ldots \qquad K_{12} = \{ P_3, P_4, P_7, P_6 \}$$

$$F_{11} = \{ K_{12}, K_{13} \} \qquad K_{13} = \{ P_{14}, P_{15}, P_{16}, P_{17}, P_{18} \}$$

Die Länge des *contourCounts*-Feldes ergibt sich für das Beispiel demnach zu elf, da der Körper aus elf Flächen besteht. Die Werte dieses Feldes entsprechen der Anzahl der Konturen pro Fläche. Das *stripCounts*-Feld gibt nun an, wie viele Koordinaten zu den jeweiligen Konturen gehören. Die Länge dieses Feldes ergibt sich aus der Anzahl der Konturen. Das *vertices*-Feld wird fortlaufend mit den entsprechenden Koordinaten aufgefüllt. Die Länge des Feldes entspricht der Summe aller Elemente der *stripCounts*, also der Anzahl der Punkte aller Konturen. Sind diese Felder ermittelt, kann man sie einem *GeometryInfo*-Objekt übergeben. Mit der Methode *getGeometryArray()* der Klasse *GeometryInfo* erhält man ein *Geometry* Objekt, welches bei der Instanziierung dem Konstruktor des *Shape3D* Objekts übergeben wird (*Abb. 5*).

$$contourCounts = \{ 2, 1, 1, 1, \ldots, 2 \}$$

$$stripCounts = \{ 4, 5, 4, 4, 4, \ldots, 4, 5 \}$$

$$vertices = \{ \underbrace{P_1, P_2, P_5, P_8}_{K_1}, \underbrace{P_9, P_{10}, P_{11}, P_{12}, P_{13}}_{K_2}, \underbrace{P_2, P_3, P_6, P_5}_{K_3}, \ldots, \underbrace{P_{14}, P_{15}, P_{16}, P_{17}, P_{18}}_{K_{13}} \}$$

```
GeometryInfo gInfo = new GeometryInfo(GeometryInfo.POLYGON_ARRAY);
gInfo.setContourCounts(contourCounts);
gInfo.setStripCounts(stripCounts);
gInfo.setCoordinates(vertices);
Shape3D shape = new Shape3D(gInfo.getGeometryArray());
```

Abb. 5: Source Code - Instanziierung eines Shape3D Objekts.

Um die Geometrie eines *Items* mit einer Brep-Repräsentation der IFC in die Java3D Form zu überführen, werden die vorgestellten drei Felder für jedes *Item* benötigt. Um

nicht im Voraus die Größe der drei benötigten Felder bestimmen zu müssen, werden dynamische Speicherstrukturen in Form einer Liste verwendet. Nach dem Auslesen der Brep-Struktur können die benötigten Felder daraus erzeugt werden. Es werden drei Listen angelegt: *contourCounts*, *stripCounts* und *vertices*. Ein *Item* besteht bei einer Brep- oder Surface-Model-Darstellung aus einem oder mehreren *IfcConnectedFaceSet* Objekten, also offenen oder geschlossenen Hüllen. Ist mehr als eine Hülle angegeben, werden die Listen für jede Hülle einfach fortgeschrieben. Der in *Abb. 6* gezeigte Algorithmus beschreibt die Erzeugung dieser Listen. Dabei wird über jede Fläche eines *IfcConnectedFaceSet* Objekts iteriert. Es wird die Anzahl der Konturen (*IfcFace.Bounds*) pro Fläche der *contourCounts*-Liste hinzugefügt. Anschließend wird über jede Kontur der Fläche iteriert. Da die *IfcFace.Bounds* nach IFC-Definition in einem Set, also einer ungeordneten Menge, angegeben werden, müssen diese sortiert werden, da die äußere Kontur der Fläche (*IfcFaceOuterBound*) vor den inneren Konturen in die Listen *stripCounts* und *vertices* eingetragen werden muss. Dafür werden zwei temporäre Listen für *stripCounts* und *vertices* notwendig. Nachdem alle Konturen der Fläche durchlaufen sind, werden die Inhalte der temporären Listen den richtigen Listen hinzugefügt. Für jede Kontur wird die Anzahl der Eckpunkte des Polygons ermittelt. Ist die Kontur vom Typ der Klasse *IfcFaceOuterBound*, wird diese Anzahl direkt an die *stripCounts*-Liste angehangen, andernfalls an die temporäre *stripCounts*-Liste. Anschließend wird mit einer Zählschleife über alle Punkte der Kontur iteriert. Ist das Attribut *Orientation* der Kontur auf „false“ gesetzt, so müssen die Punkte in entgegengesetzter Reihenfolge ausgelesen werden. Aus den Koordinaten des Polygonzugs wird ein *Point3d*-Objekt erzeugt. Analog zu der *stripCounts*-Liste wird dieser Punkt entweder der *vertices*-Liste oder der temporären *vertices*-Liste angehängt. Zum Abschluss werden äquivalente Felder mit den Inhalten der Listen erzeugt. Somit kann nun die Geometrie weiterverwendet werden, um ein *Shape3D* Objekt zu erzeugen. Das erzeugte *Shape3D* des *Items* wird dem *Object3D* hinzugefügt. Im nächsten Schritt werden Objekttransformation und Stileigenschaften ausgelesen und gesetzt.

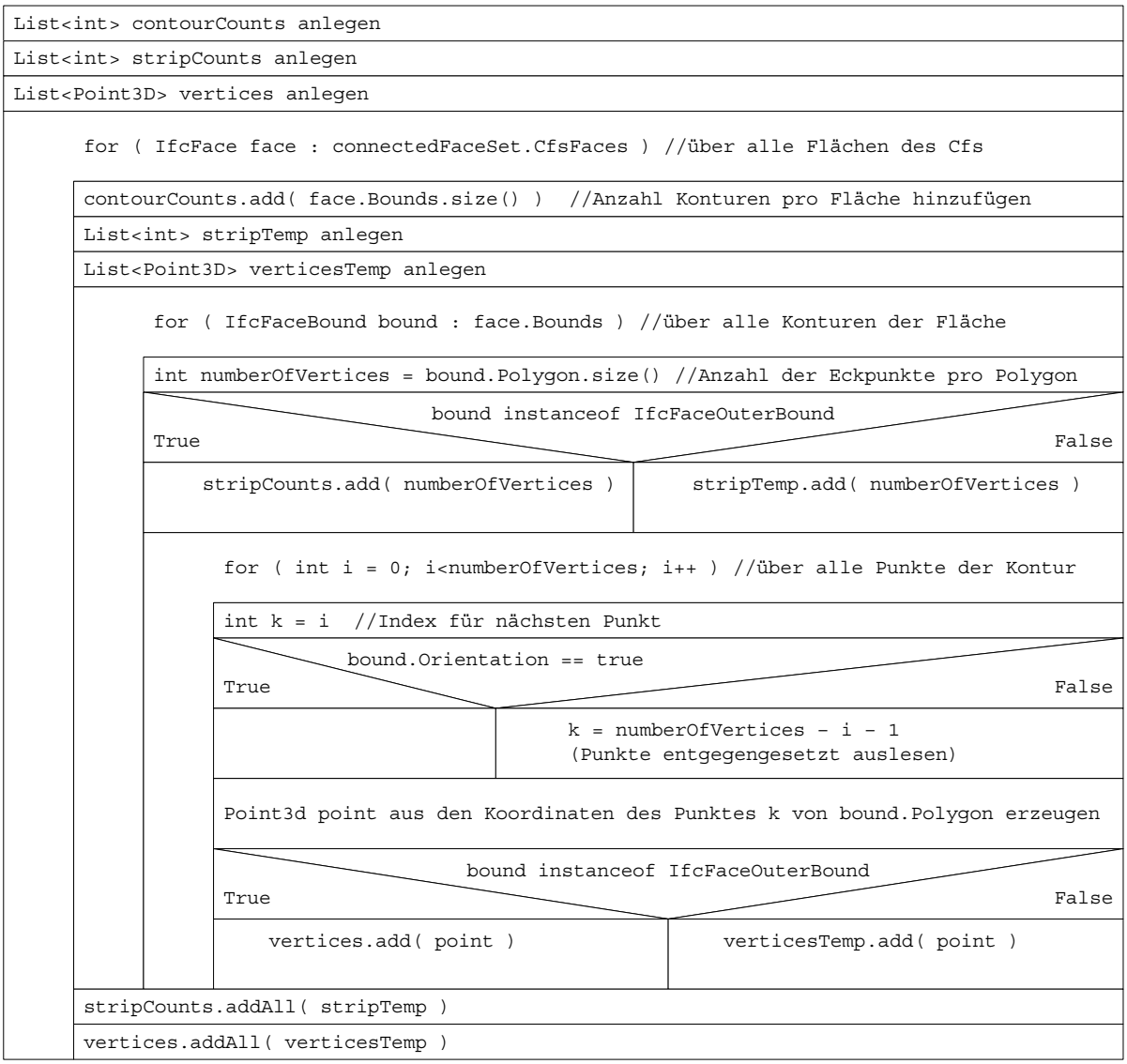

Abb. 6: Struktogramm - Überführen von IFC-Geometrie in Java3D-Geometrie.

5 Ergebnis und Ausblick

Dieser Beitrag beschreibt die Visualisierung von IFC-Objekten mittels Java3D. Er beschränkt sich dabei exemplarisch auf die Darstellung von Objekten, deren Geometrie mittels Boundary Representation (Brep) oder als Surface-Model beschrieben wird. Die notwendigen Methoden zum Überführen von Geometrie-Informationen aus den IFC in eine entsprechende Form für Java3D wurden entwickelt und gezeigt.

Der dargestellte Algorithmus ist Teil eines *IFC-Viewers*, der am Lehrstuhl Informatik im Bauwesen der Bauhaus -Universität Weimar entwickelt wird. Er ist als Open-Source-Projekt angelegt und wird unter *www.openifctools.org* veröffentlicht.

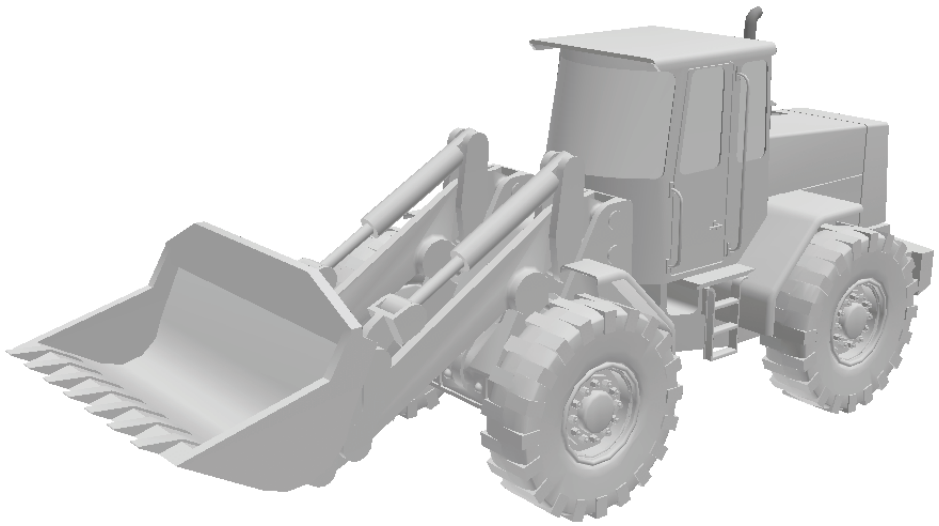

Abb. 7: Visualisierung eines Radladers.

Abbildungsverzeichnis

Literaturverzeichnis

[1] Hoffmann, Christoph M. *Geometric and Solid Modeling: An Introduction. 1989.*

[2] Liebich, Thomas. *IFC 2x3 Final Documentation. 2007.*

[3] Liebich, Thomas. *IFC 2x2 Model Implementation Guide (Version 1.7). 2004.*

[4] Bouvier, Dennis J. *Getting Started with the Java 3D API (Tutorial Version 1.5). 1999.*